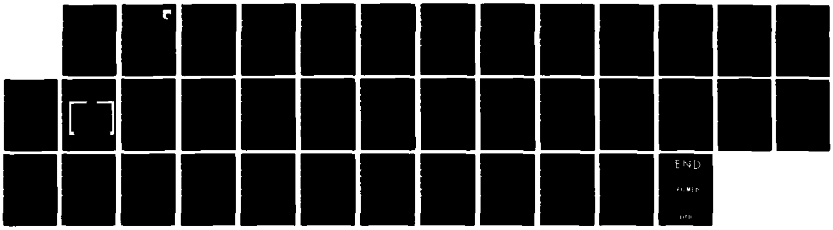
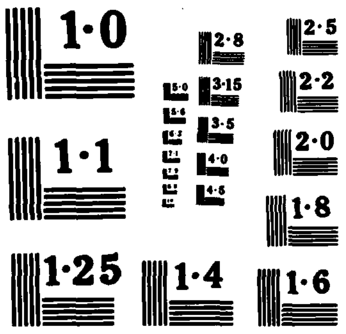
AD-A155 715 INTEGRATED SUPPORT SOFTWARE SYSTEM (ISSS)(U) GENERAL DYNAMICS FORT WORTH TX DATA SYSTEMS DIV H CONN ET AL. APR 85 FZM-7218 AFWAL-TR-85-1034 F33615-81-C-1524 1/1

UNCLASSIFIED F/G 9/2 NL




1·0
2·8
2·5
3·15
2·2
5·0
5·6
3·5
2·0
1·1
4·0
4·5
1·8
1·25
1·4
1·6

AD-A155 715

AD-A155 715

AFWAL-TR-85-1034

INTEGRATED SUPPORT SOFTWARE SYSTEM (ISSS)

Herb Conn, Fred Reagor
Scott Madaras, Hal Ferguson

GENERAL DYNAMICS CORPORATION
DATA SYSTEMS DIVISION
P. O. BOX 748, MZ 5404
FT WORTH, TEXAS 76101

APRIL 1985

FINAL REPORT FOR PERIOD JUNE 1981 - DECEMBER 1984

AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

NOTICE

This technical report has been reviewed and is approved for publication.

Mark M. Stephenson	Kenneth N. Frankovich
MARK M. STEPHENSON, 2Lt, USAF ISSS Project Engineer AFWAL/AAAF-3	KENNETH N. FRANKOVICH, Major, USAF Chief, System Evaluation Branch Avionics Laboratory

FOR THE COMMANDER

Raymond D. Bellem

RAYMOND D. BELLEM, COL, USAF
Deputy Chief
[illegible] Avionics Division
Avionics Laboratory

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) FZM-7218	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFWAL-TR-85-1034 ~~MF*23555004~~
6a. NAME OF PERFORMING ORGANIZATION General Dynamics Corp. Data Systems Division; 6b. OFFICE SYMBOL (If applicable) 81755	7a. NAME OF MONITORING ORGANIZATION Avionics Laboratory (AFWAL/AAAF) AF Wright Aeronautical Laboratories
6c. ADDRESS (City, State and ZIP Code) P. O. Box 748, MZ 5404 Ft. Worth, TX 76101	7b. ADDRESS (City, State and ZIP Code) Wright-Patterson Air Force Base, OH 45433
8a. NAME OF FUNDING/SPONSORING ORGANIZATION USAF Systems Command Aeronautical Systems Div.; 8b. OFFICE SYMBOL (If applicable) FQ8419	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-81-C-1524
8c. ADDRESS (City, State and ZIP Code) Wright-Patterson AFB, Ohio 45433	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. 62204F; PROJECT NO. 2003; TASK NO. 05; WORK UNIT NO. 24
11. TITLE (Include Security Classification) Integrated Support Software System (ISSS)	
12. PERSONAL AUTHOR(S) Reagor, Fred Madaras, Scott	
13a. TYPE OF REPORT Final; 13b. TIME COVERED FROM 6/81 TO 12/84	14. DATE OF REPORT (Yr., Mo., Day) 1985 April; 15. PAGE COUNT 35
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD 09 GROUP 02 SUB. GR. -	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Integrated Support Software System (ISSS) is an integrated software development environment that operates on a Digital Equipment Corporation (DEC) VAX 11/780 computer with the VMS operating system. The ISSS provides a UNIX-like command set, but still allows tools designed to operate on VMS to be hosted on the system. The system provides a core set of tools for software configuration management, editing, and word processing. Other tools configured for this effort are a J-73 JOVIAL compiler, 1750A tool set, JOVIAL Automated Verification System (J73AVS), Software Design and Documentation Language (SDDL), Automated Report Tracking System (ARTS), Tool Information and Presentation System (TIPS), User System Evaluation and Integration Tool (USE.IT), VAX Downloader to the AN/AYK-15A User Console (LODAYK), and X.25 Network between the VAX 11/780 and the Harris H-800. → page

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED [✓] SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Lt. Mark Stephenson	22b. TELEPHONE NUMBER (Include Area Code) (513) 255-3947; 22c. OFFICE SYMBOL AAAF-3

PREFACE

This report describes the Integrated Support Software System (ISSS) which is a collection of integrated software tools utilized in the development of software targeted for MIL-STD-1750A avionic embedded processors.

The work herein was preformed during the period 15 June 1981 to 31 December 1984 by General Dynamics Corporation Data Systems Division for the Air Force Wright Aeronautical Laboratories (AFWAL) Avionics Systems Analysis and Integration Laboratory (AVSAIL) under contract F33615-81-C-1524.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1.0 INTRODUCTION

The Integrated Support Software System (ISSS) provides the AFWAL Avionics Systems Analysis and Integration Laboratory (AVSAIL) with a software development and maintenance environment. This environment provides full life cycle support for all AVSAIL High Order Language (HOL) projects.

This document details the ideas, philosophies, and concepts involved in developing the ISSS. This final technical report presents lessons learned during the development and provides a list of ideas and recommendations for future tools and systems.

2.0 APPLICABLE DOCUMENTS

Included in this document is a brief description of the ISSS environment and its components. A detailed description of the ISSS can be found in the following ISSS documents. Documents prior to the "Phase II Interim Technical Report" are listed in that report. Only subsequent documents are listed here.

Document No.	Title	Date
FZM-7122	Phase II Interim Technical Rpt	28 March 1983
SF*23555001	System Specification for the Integrated Support Software System Type A	30 September 1982
MF*23555001	ISSS Installation/Maintenance Manual	9 October 1984
MF*23555001	ISSS Training Manual	9 October 1984
SF*21856001	ISSS Automated Report Tracking System (ARTS) System Specification Type A	10 October 1983
SF*21856001	ISSS Automated Report Tracking System (ARTS) Development Specification Type B-5	7 August 1984
SF*21856001	ISSS Automated Report Tracking System (ARTS) Product Specification Type C-5 (Parts I thru III)	14 August 1984
MF*21856001	ISSS Automated Report Tracking System User's Manual	8 August 1984

Document No.	Title	Date
SF*21656001	VAX 11/780 Downloader to the AN/AYK-15 User /Console (LODAYK) System Specification	14 October 1983
SF*21656001	VAX 11/780 Downloader to the AN/AYK-15 User Console (LODAYK) Development Specification Type B5	14 October 1983
SF*21656001	VAX 11/780 Downloader to the AN/AYK-15A User Console (LODAYK) Computer Program Product Specification Type C5	20 August 1984
PF*21656001	VAX 11/780 Downloader to the AN/AYK-15A User Console (LODAYK) Test Plan	15 August 1984
MF*21656001	VAX 11/780 Downloader to the AN/AYK-15A User Console (LODAYK) User's Manual	15 August 1984
SF*23555002	ISSS Tool Information and Presentation System Product Description Specification	15 August 1984
SF*21698002	ISSS VAX Harris X.25 Network Wright Aeronautical File Transfer (WAFT) Computer Program Development Specification Type B5	4 September 1984
SF*21698002	ISSS VAX Harris X.25 network Wright Aeronautical File Transfer (WAFT) Computer Program Product Specification Type C5	15 September 1984
MF*21698003	ISSS VAX Harris X.25 network Virtual Terminal User's Manual	15 August 1984

3.0 ISSS DESIGN PHILOSOPHY

GD proposed that the most cost-effective method of achieving the desired requirements was to use commercially available software whenever possible. In cases where no suitable software existed, GD would develop the necessary components and "integrate" them into the ISSS environment.

The resulting framework from these design decisions is shown in Figure 1.

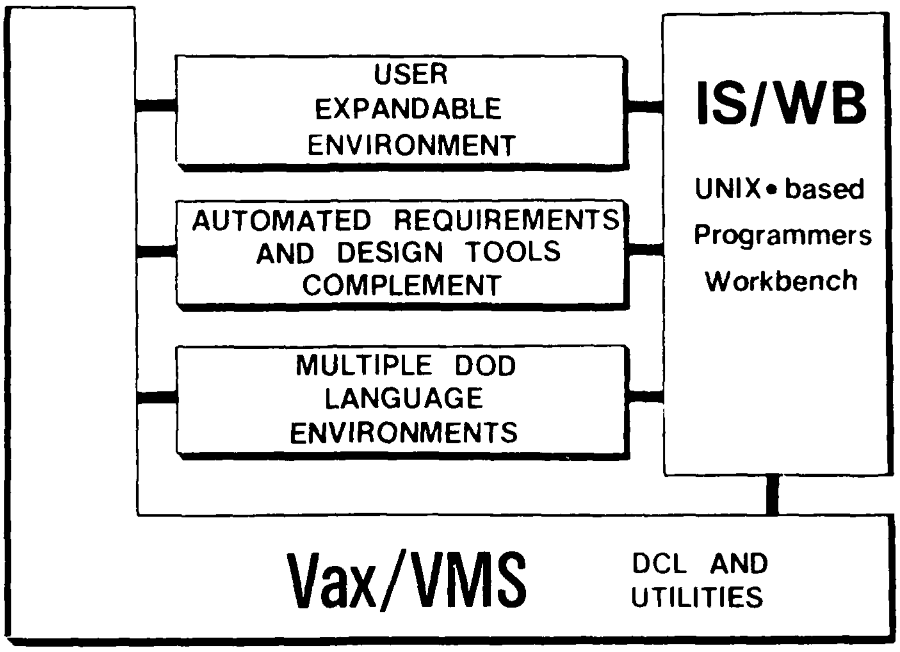

USER EXPANDABLE ENVIRONMENT
AUTOMATED REQUIREMENTS AND DESIGN TOOLS COMPLEMENT
MULTIPLE DOD LANGUAGE ENVIRONMENTS
IS/WB
UNIX• based Programmers Workbench
Vax/VMS
DCL AND UTILITIES

Figure 1. ISSS Framework

The approach was to use the facilities and tools resident within ISSS to further develop the ISSS. New tools which were built, used the existing ISSS as their development environment. For example SDDL, SCCS and MAKE were used extensively in new developments. FMS is an integral part of ARTS. The VAX help facility is an integral part of TIPS. Whenever practical, parts of ISSS were used to support the development of new features. This approach reduced development cost and promoted commonality in S/W development methodologies.

As Figure 1 illustrates, the ISSS framework has as its base, the VAX/VMS operating system. The VMS base is necessary because many of the JOVIAL/1750A tools that were available execute under the VMS operating system. Upon this base is a user-friendly software support environment contained principally in the Interactive Systems Corporation's Programmer's Workbench (IS/WB). This design allows a user to access the desired tools directly through VAX/VMS or by using the tools and facilities of the IS/Workbench. This framework provides the users of ISSS with a rich software development environment into which many software tools can be "plugged". This "plugboard" approach provides an extensible environment into which newly available tools can be easily added.

One result of the ISSS design philosophy was the discovery of the subtle distinction between "integrating" and "interlacing" tools. Not all of the various components of ISSS are truly integrated. That is, there is no common tool database and there is no integrating entity that feeds the output of one tool into another tool. The interconnection relationship of components would be more aptly described as "interlaced". This means the output of one tool is compatible with the required input of another tool when that compatibility is meaningful. This looser binding of software tools adds to the ease with which tools can be added to the environment.

Details of the component parts of ISSS are provided in the following section and in the ISSS documents listed in Section 2.

4.0 ISSS COMPONENTS

The ISSS is a VAX-based software development environment consisting of a combination of commercially available and originally created components. Full software development lifecycle support is provided by a layered architecture as shown in figure 2. The base layer is the VMS operating system overlayed by the IS/Workbench package. The top layer is a complimentary collection of software components designed to provide specific areas of support in the lifecycle development. Each of these layers is discussed in the following paragraphs.

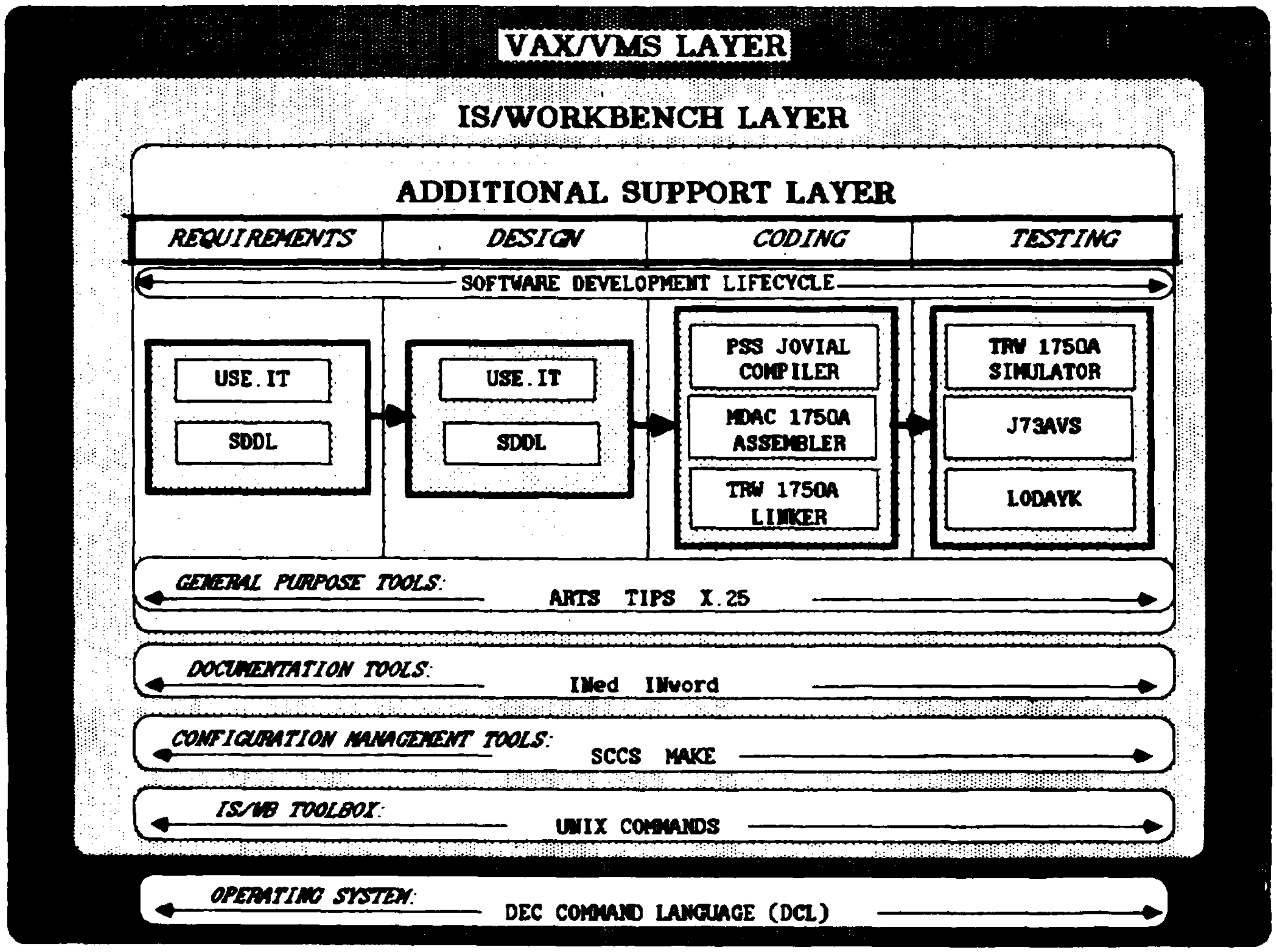

VAX/VMS LAYER
IS/WORKBENCH LAYER
ADDITIONAL SUPPORT LAYER
REQUIREMENTS
DESIGN
CODING
TESTING
SOFTWARE DEVELOPMENT LIFECYCLE
USE.IT
SDDL
USE.IT
SDDL
PSS JOVIAL COMPILER
MDAC 1750A ASSEMBLER
TRW 1750A LINKER
TRW 1750A SIMULATOR
J73AVS
LODAYK
GENERAL PURPOSE TOOLS: ARTS TIPS X.25
DOCUMENTATION TOOLS: IWed IWword
CONFIGURATION MANAGEMENT TOOLS: SCCS MAKE
IS/WB TOOLBOX: UNIX COMMANDS
OPERATING SYSTEM: DEC COMMAND LANGUAGE (DCL)

Figure 2. ISSS Software Architecture

4.1 BASE LAYER

The basic support environment for ISSS is provided by the Digital Equipment Corporation VAX Virtual Memory System (VMS) Command Language, DCL (Digital Command Language). DCL is a user friendly interface which allows file manipulation, and interactive and batch program execution and control, as well as interactive program development. Many volumes of documentation are provided by Digital Equipment Corporation on the use of DCL. All other layers of the ISSS rely on the support provided by DCL.

4.2 IS/WB LAYER

The Interactive Systems Corporation Programmer's Workbench (IS/WB) provides its users with a complete set of UNIX-like commands while running as a subsystem of VAX/VMS. The decision was made to incorporate the IS/workbench in the ISSS for two principle reasons. First, it provides its users with the full array of UNIX tools. It allows use of a very versatile command language and provides a robust array of programmer support facilities. Many of these tools were specifically called out as requirements for the ISSS. The cost effective approach was to procure these tools instead of redeveloping them. The second reason for selection of the IS/Workbench is that it "looks-like" UNIX but in fact runs under the VMS operating system. VMS was an absolute requirement for the ISSS. Many military tools, and in particular all of the JOVIAL support tools, executed under the VMS operating system, not under UNIX. The ISSS was designed to blend both, thus the choice of the IS/Workbench for VMS.

Since the IS/WB runs as a subsystem of the VAX/VMS operating system provided by DEC, full access to DEC software is always available. The Workbench command interpreter can invoke workbench utilities and also DEC utilities. Most workbench features can be directly initiated from DCL. This means that usually no additional "shell" process need be running in order to have access to ISSS.

4.2.1 Source Code Control System (SCCS)

The Source Code Control System (SCCS) is a collection of IS/WB commands that help individuals or projects control and account for changes to text files (typically, the source code and documentation of software systems). It is convenient to conceive of SCCS as a custodian of files; it allows retrieval of particular versions of the files, administers changes to them, controls updating privileges to them, and records who made each change, when and where it was made, and why. This is important in environments in which programs and documentation undergo frequent changes because of maintenance and/or enhancement work. Sometimes it is desirable to regenerate the version of a program or document as it was before changes were applied to it. Obviously, this could be done by keeping copies on paper or other media; but, this quickly becomes unmanageable and wasteful as the number of programs and documents increases. SCCS provides an attractive solution to managing these copies because it stores on disk the original file and whatever changes are made to the original file. Each set of changes is called a "delta" and only these deltas and the original file are stored.

REFERENCE MANUAL:
IS/Workbench for VAX/VMS Programmer's Guide

4.2.1.1 Commands

The following is a summary of all SCCS commands and their major functions:

get......Retrieves versions of SCCS files.

unget....Negates the action of the previous get.

delta....Applies changes (deltas) to the text of SCCS files, i.e., creates new versions.

admin....Creates SCCS files and applies changes to parameters of SCCS files.

prs......Prints information stored in an SCCS file according to user specified format.

inhelp...Gives explanations of diagnostic messages.

rmdel....Removes a delta from an SCCS file; allows the removal of deltas that were created by mistake.

cdc......Changes the commentary associated with a delta.

what.....Searches any IS/WB file(s) for all occurrences of a special pattern specified by the SCCS identification keywords and prints what follows the pattern.

sccsdif..Shows the differences between any two versions of an SCCS file.

comb.....Combines two or more consecutive deltas of an SCCS file into a single delta; often reduces the size of the SCCS file.

val......Validates an SCCS file.

NOTE: The "man" command can be used to obtain help information on these and other workbench commands.

4.2.2 MAKE (Automated Program Build Facility)

The MAKE program mechanizes many of the activities of program development (i.e. compiling, linking, etc.) and maintenance. If the information on inter-file dependencies and command sequences is stored in a file, a single command is frequently sufficient to update the necessary files, regardless of the number of files that have been changed since MAKE was last invoked.

A common practice is to divide large programs into smaller, more manageable pieces which may require quite different treatments. Related maintenance activities involve running complicated test scripts and installing validated modules. However, a programmer can forget which files depend on other files or which files have been modified recently and the exact sequence of operations needed to make or exercise a new version of the program. Forgetting to compile a routine that has been changed or that uses changed declarations can be disastrous for program execution; but recompiling every module for guaranteed execution can be very time consuming, as well as a wasteful use of computer resources. The MAKE program mechanizes these management activities of program development and maintenance. If the information on inter-file dependencies and command sequences is stored in a file, a frequently sufficient command to update the necessary files is: $make [carriage return]. In most cases, the description file is easy to write and changes infrequently. It is usually easier to type the make command than to issue even one of the needed operations, so the typical cycle of program development operations becomes

think -- edit -- make -- test ...

4.2.3 INword (INteractive Word Processor)

INword is a word processing facility that provides the capability to create, format, proof, revise, and print files (documents). The document is created and edited using the INed text editor with the formatting instructions embedded in the document.

4.2.4 Interactive Text Editor (INed)

INed is INTERACTIVE SYSTEMS screen text editor which provides a two-dimensional window into a text file. INed's command language features function keys for opening, deleting, and moving files, characters, and rectangular portions of text on the screen. Interactive text processing includes paragraph file and right margin justification and may be expanded to include additional user or system-provided programs that are invoked interactively from within the editor. The screen may be divided into several windows to simultaneously edit one or more files.

System functions, other than editor commands, can be invoked from within the editor, placing their results in the editor. For example, a formatted list of files can be produced for a document by entering the editor, writing the document up to the place where the file list is desired, and invoking the "ls" command, which will place a directory listing in the document. Once the file list is in the document, it can be formatted using normal editing commands.

4.2.5 Other Commands

The UNIX-like command set of the IS/WB provides over two hundred additional completely integrated commands, each of which is discussed in detail in the IS/Workbench User's Manual and can be displayed using the workbench "man" command.

Of particular interest are the concepts of filters, redirection, and pipes. Some of the UNIX commands act as filters because input data is converted by the command before being output. Redirection allows the standard I/O defaults to be changed during a particular command sequence. A pipe allows two or more cooperating processes to pass data between them without the need for temporary files.

User communications are enhanced by the INmail and INnet features of the workbench. INmail is an electronic message sending and receiving facility. Each user on the system has a private mailbox for receiving mail from other users. INnet is a group of facilities that provides for transfer of mail and files over a network to other VAX computers equipped with INet.

4.3 TOP LAYER SUPPORT

The top layer is provided by a complimentary set of software tools designed to provide support in specific areas of the lifecycle of software development. Some tools were purchased from commercial sources or provided as Government Furnished Equipment (GFE). Others were developed during the ISSS project by General Dynamics or Systran. These are discussed in more detail in the following paragraphs.

4.3.1 Commercial/GFE Software

Tools purchased commercially or provided as GFE are 1) User System Evaluation and Integration Tool (USE.IT), 2) Software Design and Documentation Language (SDDL), 3) PSS JOVIAL compiler, 4) MIL-STD-1750A tools and 5) the J73 Automatic Verification System (J73AVS). These tools are discussed further in the following paragraphs.

4.3.1.1 User System Evaluation and Integration Tool (USE.IT)

USE.IT automates a mathematically based methodology developed by Higher Order Software, Inc. (HOS) in Cambridge, MA. USE.IT was designed to provide a complete environment for the development of software by verifying the consistency and logical completeness of the design at the specification level. The HOS methodology forces the generation of specifications that are complete and consistent. Frequently, it is possible to automatically produce source code from the specifications. Current language targets include Pascal, FORTRAN and COBOL.

The USE.IT System is made up of four main components:

USER INTERFACE

The User Interface is a menu-driven front end which allows easy interaction with the other main components of USE.IT. You may choose between command options, enacting them with one or two keystokes. The User Interface also provides a very convenient listing of models within a system and the development status of each model.

A great aid in managing the diverse elements of a system under development, the User Interface eliminates much confusion and provides a straight-forward approach to USE.IT.

AXES

Using the USE.IT graphics editor, you may construct control maps for processes in a system. Each control map constitutes a model. Graphical AXES is the highly structured hierarchical language with which you may specify all functions in the model using three basic units:

o data types (input/output variables)

o functions

o control structures

ANALYZER

The Analyzer performs static analysis on partial or completed models. It checks to ensure that all parts of the model are internally consistent, and checks all interfaces for correctness and completeness. No model may be implemented in a program unless it passes the Analyzer.

RESOURCE ALLOCATION TOOL

The Resource Allocation Tool (RAT) automatically generates programs from successfully analyzed models. The RAT ensures that the implementation is strictly consistent with the definition as constructed in the models. The RAT eliminates the need for error-prone manual programming, permits simulation, and makes rapid prototyping possible. Most important, it enables you to reconfigure to any other language or machine without modifying the requirements set forth in AXES.

4.3.1.2 Software Design and Documentation Language (SDDL)

The Software Design and Documentation Language (SDDL) is a software support tool used to partially automate the generation and checking of a software design document (SDD). The objective of SDDL is to support the design and documentation of complex software by providing (1) a text processor which can convert design specifications into an intelligible, informative, machine reproducible document, (2) a design and documentation language with forms and syntax that are simple, unrestrictive, and communicative, and (3) a methodology for effective use of the language and text processor. The text processor can format documents, summarize design information in the form of reports and act on various user-controlled directives.

SDDL is accessed interactively, but is a batch oriented process. It is used to periodically generate a SDD which will reflect the current status of the design process. Information may be generated on variable usage, program modules, changes to programs, management information and many other states of the design as desired by the project team members. A programming language's structure and keywords may even be modeled to an extent that very little effort would be required to change the final SDD into source code.

Utilization of SDDL is accomplished through a language syntax that is simple and flexible. The SDDL processor reads a text file phrased in the language, then reformats the file by providing indentation, control flow lines, and user specified cross reference tables. The printed SDD contains the reformatted input, a table of contents, and module hierarchy reports. The indentation can be modeled by using the structural keywords of a specific language. The exploitation of these and other features of SDDL provides a vehicle for establishing standards and conventions in the design process.

SDDL is a product of SAI, La Jolla, Ca.

4.3.1.3 PSS JOVIAL Compiler

The Proprietary Software Systems (PSS) JOVIAL Compiler currently hosted on the ISSS is the one which is available as GFE (Government Furnished Equipment) from the Embedded Computer Systems Program Office (ECSPO) at Wright Patterson AFB.

This compiles MIL-STD-1589B JOVIAL into VAX-11 object code or into MIL-STD 1750A assembly language. The 1750A object code produced by the compiler is compatible with the ECSPO tool set. (1) the McDonnell Douglas 1750A macro preprocessor, and assembler (2) the TRW linker and simulator. These tools are outlined in the following paragraphs.

4.3.1.4 MIL-STD-1750A Tools

The ECSPO toolset consists of the McDonnell Douglas macro librarian, macro preprocessor, assembler, and reformatter, and the TRW linker and simulator. The macro librarian maintains (adds, deletes, replaces) macros in a library; the macro preprocessor expands all macros into assembly language. The assembler translates assembly language code into MIL-STD-1750A object code. The reformatter takes the object code output from the assembler and reformats it for input to the ALINKS linker. The ALINKS linker combines object modules from the JOVIAL compiler and reformatted object modules from the assembler into a executable image. A symbol table can be generated for input to the TRW simulator.

The TRW simulator allows the 1750A instructions to be executed in software on the VAX 11/780. Its inputs are the executable image and symbol table from the ALINKS linker. The executable image from the ALINKS Linker can be downloaded from the VAX 11/780 through an RS-232 link to an AN/AYK-15A 1750A embedded computer using LODAYK.

4.3.1.5 J73 Automated Verification System (J73AVS)

The J73AVS (J73 Automated Verification System), a product of General Research Corporation, is a JOVIAL testing tool. It can interactively check syntax and data flow. It can generate statistics on the source code, and it can instrument source code for testing (various tracings, timings, etc.).

J73AVS is a software tool that analyzes either the MIL-STD-1589A or 1589B dialects of JOVIAL/J73. The goals of the system are to provide automated assistance for program development, debugging, testing, retesting, maintenance, and software documentation for JOVIAL J73 programs. The user's JOVIAL source code is instrumented and used by J73AVS to direct the type of analysis to be performed.

4.3.2 General Dynamics/Systran Developed Software

Tools developed by General Dynamics or Systran for the ISSS contract are the 1) Tool Information and Presentation System (TIPS), 2) Automated Report Tracking System (ARTS), 3) Load File to AYK-15 (LODAYK), and 4) X.25 Network Software. These are discussed further in the following paragraphs.

4.3.2.1 Tool Information Presentation System (TIPS)

TIPS is an on-line help facility for ISSS. For each software tool available under ISSS, TIPS provides a description of the tool, an explanation of its invocation and the reference materials available for further information. TIPS is based upon the VAX VMS "help" facility making it easily expandable.

When a user logs onto the VAX 11/780, the following message will be displayed:

```
        WELCOME TO ISSS.
FOR FURTHER INFORMATION, TYPE ISSS
```

When ISSS is typed, TIPS is invoked. TIPS can be invoked at any time during the login session by typing ISSS [carriage return] whenever the Digital Command Language (DCL) prompt ($) is displayed. Once the user has entered TIPS, the desired keyword for ISSS tool information may be chosen from a list of keywords displayed on the screen by typing the keyword followed by a carriage return.

The following is a list of keywords used by TIPS.

ARTS	MAKE	USE.IT
INed	MS1750	WORKBENCH
INword	SCCS	X.25_NETWORK
J73AVS	SDDL	
JOVIAL	TOOLS	
LODAYK	TRAINING_MANUAL	

4.3.2.2 Automated Report Tracking System (ARTS)

ARTS is a menu-driven program specifically developed for ISSS that allows the user to select one of the following functions: input of reports, modification of reports, and outputting lists of selected reports. The reports that ARTS will process are: Problem Reports (PR's), Engineering Change Proposals (ECP's), Emergency Configuration Change Notices (ECCN's), Specification Change Notices (SCN's), and Document Change Notices (DCN's).

The ARTS program is written in VAX-11 Pascal. The program also uses a screen-formatting package, FMS, for all menu and report displays. Since FMS employs direct cursor addressing, ARTS may only be run using a VT100 or VT100 compatible terminal. ARTS also uses some of the DCL and IS/WB commands to perform text processing and file manipulation directly from the program.

4.3.2.2.1 Functional Descriptions

This section identifies and describes the allocation of functions and tasks to be performed by the individual computer subprograms. The following is a list of the ARTS source files and their use.

1) ARTS.PAS - This file contains the initialization and cleanup routines.

2) ARTSDEF.PAS - This file contains all of the global constants and type definitions used by ARTS

3) CI.PAS - This file contains the routines to handle Configuration Item Descriptions.

4) CMOMENU.PAS - This file contains the routines to select the next menu or report to be displayed based on user input.

5) DCN.PAS - This file contains the routines to handle Document Change Notices.

6) DOCS.PAS - This file contains the document configuration item specific routines.

7) ECCN.PAS - This file contains the routines to handle Emergency Configuration Change Notices.

8) ECP.PAS - This file contains the routines to handle Engineering Change Proposals.

9) HW.PAS - This file contains the hardware configuration item specific routines.

10) MENUHAN.PAS - This file contains general purpose menu handling routines to display menus, perform scrolling of menu choices, and to obtain user selections.

11) PR.PAS - This file contains the general routines to handle Problem Reports.

12) PR1.PAS - This file contains the routines specific to the top half of the PR form.

13) PR2.PAS - This file contains the routines specific to the bottom half of the PR form.

14) REPHAN.PAS - This file contains general purpose report handling routines.

15) REPUTIL.PAS - This file contains general report utilities.

16) SCN.PAS - This file contains the routines to handle Specification Change Notices.

17) SORT.PAS - This file contains the sorting routines used for log generating

18) SW.PAS - This file contains the software configuration item specific routines.

4.3.2.3 Load File to AYK-15 (LODAYK)

LODAYK is used to 1) download files from the VAX to an AN/AYK-15A processor and 2) to allow the user to execute AN/AYK-15A processor User Console (UC) commands from any VAX computer terminal.

LODAYK is written in VAX-11 FORTRAN and utilizes the FORTRAN accessible VAX System Service Routine "QIO" for the I/O operations between the VAX and the AN/AYK-15A User Console. Also LODAYK converts a file, output by the Avionics Linker (ALINKS), to a character oriented file suitable for transmission over an RS-232C channel.

4.3.2.3.1 Function Allocation Description

LODAYK is comprised of five major functions: the main routine, an input file reader/converter, output handler, input handlers and command handler routines.

The main program, called LODAYK, controls the assigning and relinquishing of I/O channels, performs variable initialization and interacts with the user to initiate commands for the AN/AYK-15A User Console.

The input file reader/converter is a subroutine called REFORMAT_VAX_FILE. It opens the specified input file, reads in the data, converts the data to ASCII Hex characters, and writes it out to a temporary file to be downloaded.

The output handler is a subroutine called CIUOUT. It outputs character string data to the User Console.

There are two input handler subroutines called CIUIN and CIUIN2. Both subroutines collect character string data from the User Console. CIUIN2 is used in the cases where multiple buffers are needed to accept larger amounts of User Console output.

The command handler subroutines are in the files CIUSUB and CIUXFER. Each subroutine handles the processing of a particular command.

4.3.2.4 X.25 Network

The X.25 networking system is a Packet Switching Network (PSN) between the VAX 11/780 computer system and the Harris 800 systems located in AVSAIL. The PSN system permits a user of either VAX or Harris systems to sign on as a virtual terminal user to the alternate system. This network fully conforms to the CCITT standards for X.25 communications.

The X.25 network supplies two capabilities to the users of the VAX and Harris systems. The PAD command provides virtual terminal capabilities to terminals on either the VAX or Harris systems. The WAFT command provides a file transfer capability to terminals on the VAX system. The WAFT command is available to Harris users by connecting to the VAX through the PAD.

4.3.2.4.1 PAD

The PAD (Packet Assembly/Disassembly) command permits a user of either the VAX or Harris systems to sign on as a virtual terminal user to the alternate system. After signing on as a virtual terminal, the user's terminal appears to be a teletype to the other machine. The user is able to carry out any function which can be done on a teletype (dumb terminal) on the other machine. Once a user has entered the network and connected to the target system, all commands are single line teletype mode entry. These commands must be in the format of the target system.

Individual characters typed at the terminal are assembled into packets via the (PAD) facility: then they are shipped through the network to the remote computer; and, finally, the packets are disassembled for storage at the remote site. The X.25 virtual terminal capability on the AVSAIL VAX/Harris systems is implemented through the host based PAD resident on each system.

4.3.2.4.2 Wright Aeronautical File Transfer (WAFT)

The Packet Switching Network (PSN) system permits a user of the VAX system to transfer text files to or from the Harris system. The Wright Aeronautical File transfer (WAFT) program carries out this function.

The VAX WAFT programs is initiated by the command "WAFT". After executing WAFT, the user is prompted for several items of information including the Harris logon line, direction of transfer, VAX filename and Harris areaname. The program then transfers the file according to the user's input. The transfer files must be ASCII files.

5.0 ISSS IN THE SOFTWARE COMMUNITY

The Phase II interim report of March 1983 discussed the briefings and demonstrations given to government agencies and internal to General Dynamics. Since that time, several of these groups have obtained components of the ISSS for their project development work. The use of the ISSS by these groups is discussed in the following paragraphs.

5.1 USE OF ISSS AT GOVERNMENT AIR LOGISTICS CENTERS (ALCs)

The Air Force Wright Aeronautical Laboratory has supplied a copy of the IS/WB portion of ISSS to the Ogden, Utah and Oklahoma City, Oklahoma ALCs for beta test in an effort to standardize development environments in the Air Force. The informal response has been good. As of this date, no formal report from the ALCs is available for inclusion in this report.

5.2 USE OF ISSS INTERNAL TO GENERAL DYNAMICS

The IS/WB has been purchased for use within the Fort Worth and Data Systems Divisions of General Dynamics in the Production Digital Flight Controls (PDFC), Electrical Harness Data Systems (EHDS), Defense Mapping Agency (DMA) Modern Programming Environment (MPE) contract, and the Advanced Technology Group.

5.2.1 Use by Production Digital Flight Controls (PDFC)

The IS/WB has been purchased for use by the PDFC group initially based on the improved text editor and word processing capability. Other features of the IS/WB will be evaluated for use by the group after the IS/WB software is received in-house. Configuration control has been established for several years, thus there was no crucial need for SCCS. The current development utilizes J73 and 1750A running under VMS. Thus the IS/WB will be quite compatible with the current working configuration.

5.2.2 Electrical Harness Data Systems (EHDS)

This group was not mentioned in the interim Phase II Technical Report, but they had a requirement for strong configuration control and powerful file handling capability, such as that provided by SCCS and UNIX-like command structure (cat, grep, etc.). They found out about IS/WB and requested a demonstration. Subsequently, IS/WB was purchased and has been in operation since May 1984. This is not a software development operation, so the primary need is still in configuration control, but the group is pleased with the operation and use is to be continued indefinitely.

5.2.3 Defense Mapping Agency (DMA) Contract

The Advanced Technology Department of Data Systems Division is currently on contract to provide a Modern Programming Environment (MPE) for the DMA. The DMA is interested in a VAX-based development environment that would support VMS (because of the number of tools that are available) with the power of UNIX. This is the primary reason why the IS/WB was chosen. In addition, the SDDL, and USE.IT tools are included for their ability to aid in requirements and design definition. In particular, SDDL was chosen for its flexibility, ease of use, and maturity. The tool is flexible because it can be used across multiple languages. JOVIAL, however, is not one of the languages used, so there will be no need for J73 or MIL-STD-1750A capabilities.

5.2.4 Advanced Technology

The Advanced Technology Department has utilized the commercially available components of ISSS on the Fort Worth Division Engineering VAX for about two years. The editor and SDDL have had particularly heavy use in previous projects.

The IS/WB is being installed on the new Advanced Technology Department's VAX, and is intended for use by the development group in fulfillment of current and future contracts.

6.0 LESSONS LEARNED

This section details the lessons learned by General Dynamics during the development of the ISSS. These details only cover experiences directly related to the development of the ISSS environment. Section 7.0 takes these experiences and adds in new insight as well as information about new tools and develops a series of recommendations for an improved software development environment.

6.1 USING ISSS AS A SOFTWARE DEVELOPMENT ENVIRONMENT

During the development of the Automated Report Tracking System (ARTS), some valuable insights into using ISSS as a software development environment were made. ISSS tools were utilized in the development effort, in the documentation effort, and as a functioning part of the program. ARTS was written in Pascal using FMS and IS/WB tools consistent with the ISSS philosophy. A detailed discussion of each of these efforts is given in the following paragraphs.

6.1.1 SDDL as a Documentation Tool

The Software Design and Documentation Language (SDDL) played a major role in the documentation of the ARTS and LODAYK programs. The program flow information provided by SDDL listings eliminated the need for conventional flow charts. Further, a functional flow diagram, which is required in a product specification, is automatically generated by SDDL.

Information was gained on how to use SDDL with Pascal to gain the most useful items of information. Minor structural changes to the Pascal source code were necessary so SDDL could handle subroutine nesting, subroutine invocation and certain Pascal control structures. Special comment sections describing the constants, type definitions, and variables used by ARTS were necessary for SDDL to create detailed cross reference listings of all identifiers. It should be noted that these structure changes to the Pascal code were of the form of comments and/or indentation and had no effect whatsoever on the functionality of the code itself. For details regarding the use of SDDL for ARTS, reference section 3.5.9 of the ARTS Production Specification.

6.1.2 Configuration Management of the Development Effort

Some key features of the ISSS environment are Configuration Management and Configuration Control of source code and documentation. It was necessary to learn how to use the available automated CM tools and to apply the tools properly. Further, a methodology was developed for the organization and layout of the source code modules and the use of various disk directories. This established the overall organizational methodology. These ideas will be discussed in Section 7.3 as general project development methodologies.

6.1.3 Impact of ISSS Utilities on the Functionality of the System

The purpose of a software development environment is to provide utilities that aid the development effort. An indication of the power and flexibility of ISSS is that the tools and utilities can be used as part of the functionality of a software system. These utilities simplify the development effort because large amounts of code can be replaced by a few commands. The resulting product becomes more flexible and reduces the development cost. This concept of using ISSS utilities to perform some of the functions of a system was demonstrated in the Automated Report Tracking System (ARTS).

The basic functions of ARTS include entering reports, modifying reports, and producing listings (logs) summarizing the entered reports. The nature of this program dictates the need for storage and retrieval of data records. Although this would typically be performed by some type of data base manager, ARTS merely uses text files to store the records. ISSS utilities are called to search for records, modify records, or perform formatting functions on the data. A distinct advantage of this method is that information can be retrieved from the master report file without actually running the ARTS program. Further, data can be extracted based upon the data in any field or fields at any time and in ways not foreseen at the time of the requirements specification of ARTS.

6.2 USER RESPONSE

Section 5 discussed the various installations where ISSS resides. These installations have provided GD with good feedback on the acceptance and use of the ISSS environment. This has been especially useful in those instances where software developers were using the facilities of ISSS. The following paragraphs present their observations and impressions of these tools.

The most difficult aspect of introducing new technology is making people aware that it exists, and getting them to use it. Towards that end, GD provided the various agencies using ISSS with training courses that made people aware of what was available and how it could help them. The training classes offered "hands-on" lab experience so that the "fear-of-the-unknown" aspect of new tools would be diminished. It appeared that the training classes were well accepted. They did initiate people to the tools and utilities. Exposure to new technology and training in it is a very underrated aspect of new tool development. Training is crucial to the infusion of new technology and in increasing productivity. This fact cannot be overlooked, and all too often is.

Another important aspect of new tool development, besides knowing how to use the tool, is knowing when to use it. This leads to the notions of methodologies and scenarios. A tool can be the most powerful one available and people can be taught how to use it, but it won't be used if people can't see how to apply the tool to their problem. Such a situation as this occurred with the Source Code Control System (SCCS) tools. SCCS is a set of tools that provides configuration control of text files (source code, documents, etc.) SCCS was installed at Hill AFB, Ogden, Utah where General Dynamics trained the personnel in the use of the tools. However, when the programmers attempted to use SCCS on a development project they experienced difficulties. The existing system did not organize their files and directories in a manner that worked well with SCCS. An example of a development methodology and a scenario that give examples and suggestions of appropriate file and directory organization might have prevented this problem.

Both VMS and the IS/WB provide a powerful series of commands and equally powerful command language interpreters. However, they also provide separate and distinct user interfaces. Although the differences are mostly syntactical, users expressed the desire for a common user interface. In addition, the ability to mix VMS and IS/WB commands would help to alleviate the awareness of two distinct command language interpreters. Further, the sometimes cryptic nature of the various IS/WB commands (e.g. "cat" a file) has been a topic of slight annoyance to most all non-UNIX users.

Although Interactive Systems Corporation provides a lot of documentation for their system, the descriptions and examples of the various commands tend to be somewhat cryptic. This is not true of their documentation for INed and INword though. To help meet the user needs of good documentation, GD has produced both on-line documentation (TIPS) and written documentation (ISSS Training Manual and the ISSS Installation and Maintenance Manual) to assist the user. The Tool Information and Presentation System (TIPS) provides the user with on-line information regarding the various component pieces of ISSS. It also identifies specific written documentation that the user should consult for further information. The ISSS Training Manual is an introduction and instructional aid for ISSS users. This manual provides an explanation of the purpose of the ISSS components and how they relate to each other. Finally, the ISSS Installation and Maintenance Manual is a collection of all the necessary information to install, maintain, and sometimes enhance the software tools of ISSS. Together, TIPS and these two documents provide an overview picture of the ISSS and detailed information on its various components.

6.3 Experience with USE.IT

The USE.IT tool can be applied to 1) software development from requirements through code generation, 2) requirements only, or 3) rapid prototyping. But experience suggests that the best potential use of the tool is in the area of requirements definition.

The reasons for this conclusion are as follows. 1) Although the USE.IT tool is maturing, there are still some problems with the FORTRAN code generator when complex specifications are used. 2) A tremendous amount of specification detail is required in order to produce code automatically. 3) On a large project, the architecture of the design may not match the structure of the requirements definition. Thus the USE.IT tool could lead a designer into a less than optimal design configuration. 4) Only basic support libraries exist, making system functions, especially screen formatting, very difficult. 5) Primitive definitions, essential for the proper operation of USE.IT applications, are difficult to define.

The reasons why USE.IT is a good tool for requirements definition lie in the graphics capability of the tool and in the ability to get fast hardcopy of the graphics on a plotter. The analysis tool can check for missing parts of the specification and notify the user of inconsistencies. Also, textual comments can be added using the documentor utility. Thus, all the basics of a good requirements specification tool are met.

7.0 OBSERVATIONS/RECOMMENDATIONS

This section extends the lessons learned during the ISSS project into a series of observations and recommendations for additional enhancements to a software development environment. These observations and recommendations are presented in the following sections. The first section presents the general attributes for a desired environment. The next two subsections discuss the need for a more integrated environment. The final section looks at some new or additional tools and how they could enhance the ISSS environment.

7.1 GENERAL REQUIREMENTS FOR A SOFTWARE DEVELOPMENT ENVIRONMENT

Software development is very complex and difficult to manage. In an effort to ease the burden of software development, various software tools have been proposed and built. As stand-alone elements, these tools provide a narrow scope of help to the development process. A software development environment is a collection of tools whose union provides support across the full software development lifecycle.

A problem exists for environments that also exists for programming languages, documentation, etc. There is little standardization within one branch of the armed forces let alone DoD wide (Although Ada is an attempt to solve that problem for the programming language aspect). A major reason that environments haven't been standardized is that they have been too narrow in scope and too tightly bound to site-specific needs.

Additional requirements for software development environments include transparency, expandability, and modularity. The existence of the environment should be transparent to the user and the tools should be available at all times. An environment should be expandable so that new tools can be added to the system and unsuitable tools can be replaced or deleted without affecting the rest of the system. To help achieve the goals of transparency and expandability, the environment should also be modular. That is, it should not be one monolithic entity in the background, tying up resources and generally tying up a system. It should be an ordered collection of compatible tools that are interlaced. Interlaced means that the tools are loosely bound together making the transition to new tool technology an easy task. The term interlaced is used instead of integrated. Integrated tools implies a stronger interleaving of tools and the existence of dependencies between the tools. This makes the addition/subtraction of tools unwieldy.

These basic requirements outline the specifications for a "generic" software development environment that is both language independent and machine independent. The following sections give specific details of this kind of software development environment.

7.2 FULL SOFTWARE PRODUCT SUPPORT

As mentioned earlier, single tools usually provide support of a single phase of the software lifecycle. By combining a series of carefully chosen tools into an environment, the full software development lifecycle can be supported. However, this is still an incomplete support package. Ideally, an environment should provide full software product support, as defined by Richard C. Gunther in Management Methodology for Software Product Engineering.

Mr. Gunther states that software product support includes software lifecycle support, but adds the concepts of software planning, estimating, documentation, quality assurance, quality control, training, distribution, and maintenance. Some of this support can be automated using available tools, while others are implemented with good software methodology practices and software management techniques. Within a particular organization, these methodologies and techniques should be uniformly and consistently applied. Further, all such methodologies and techniques should be fully documented for each software product.

7.3 SOFTWARE DEVELOPMENT NEEDS MORE THAN TOOLS

A good software development environment can provide much assistance to the software development effort. However, the existence of these tools does not mean that they will be used properly or used at all. Non-tool related components of an environment are needed to complete the software development picture. These additional components are software development training, documentation, methodologies and scenarios.

7.3.1 Training

As mentioned previously, training is frequently neglected when a new tool becomes available. Users need to know what tools are available. They need to be sufficiently trained in the use of the tools, so that using them won't stand in the way of productivity. Just as importantly, a user needs to know when to use the tools. This last aspect relates to the remaining ideas of this subsection: documentation, methodology and scenarios.

7.3.2 Documentation

The developers of an environment should provide an abundance of useful documentation, both on-line and in written manuals. On-line information should provide a user with a view of the environment and top level details of each component. It should also reference specific manuals that can be consulted for additional information. On-line "help" information giving the details of how to invoke the various tools and options available should also be provided.

Detailed documentation of the various components of the environment, as well as the environment as a whole, should be provided. This includes the development documentation and user documentation. Development documentation can be found in the following manuals: Requirements Specification, Development Specification, Product Specification, Installation Instructions, Maintenance Instructions and Enhancement Instructions (how to add/subtract tools from the system). User documentation includes user's manuals and training manuals for each individual tool and for the system as a whole. The training manuals should include examples as well as training exercises for hands-on experience. The training manual could be an interactive computer-based instruction course. Finally, there should be documents giving realistic examples of software development methodologies and scenarios.

7.3.3 Methodologies

A methodology is a set of methods, rules, guidelines, and procedures to follow for software development. It is not the purpose of this report to dictate which methodologies should be adhered to during software development. However, it is imperative that a set of procedures be established and followed.

Some of these may be global in scope, such as file and directory organization, configuration management and control schemes, and documentation standards. Other procedures may be localized, i.e. site dependent, language dependent, or machine dependent. The same set of local methodologies may not be appropriate for all software developments, but the use of some set of guidelines and procedures should be enforced. These local procedures should be uniform and consistently applied. Further, they must be fully documented as part of the system development effort. By forcing the use and the documentation of these guidelines, the tendency to skip the guidelines due to time/budget constraints will be diminished.

Methodologies should be outlined for: 1) documentation standards, 2) use of CM tools in documentation, 3) directory structure, 4) use of CM in software development, 5) testing criteria and CM, 6) software deliveries, and 7) installation and maintenance. Further guidelines should be established for full software product support, as outlined in Section 7.2.

7.3.4 Scenarios

A scenario is an outline for, or an example of a sequence of events. As applied to software development, scenarios constitute an example of a sequence of events in software development. This includes the tracking of a software product development throughout the entire software lifecycle. It includes giving examples of when to use specific tools and how specific methodologies should be applied. To help make a software development environment complete, different scenarios should be developed, documented, and presented to the eventual end user of the system. This information will help the user to know how and when to apply certain tools. It gives the user a "template" upon which to structure his individual application.

7.4 TOOLS FOR ENHANCING AN ENVIRONMENT

During the Requirements Analysis Phase of ISSS, many tools were reviewed and evaluated. The best and most cost-effective set of tools were combined to form the ISSS environment. Since that time, new tools have become available that could offer enhancements to ISSS. Also, some tools not yet developed for the VAX have been identified. The next sections present some of these tools and ideas, and what advantages they offer to software developers.

7.4.1 Problem Statement Language/Problem Statement Analyzer (PSL/PSA)

PSL/PSA is a requirements definition language system. This system consists of several tools: PSL (Problem Statement Language), a language used to express system requirements and specifications; PSA (Problem Statement Analyzer), which generates 48 different reports which display and analyze the requirements and specifications in different ways; a Query System, a Document Generator, an Algorithm Development Language Processor, various utility programs and a Data Base Manager that ties the system together. This set of tools provides support for the requirements analysis phase of software development. PSL/PSA is available from ISDOS corporation in Ann Arbor, MI.

7.4.2 Change and Configuration Control (CCC)

The Change and Configuration Control (CCC) System is a configuration management tool by Softool Corporation in Goleta, CA. It provides the same capabilities as SCCS with the addition of some other useful features. Particularly, CCC handles the whole project: source code, object code, executable, and documentation. Each project exists as an entity of its own. Information is organized in a hierarchical format that allows easy identification of logical subsystems. CCC also provides a more detailed protection mechanism, a more friendly user interface, and a complete recovery system. These additional features provide a well-balanced set of configuration control abilities.

7.4.3 Ten Plus

Ten Plus is a user-friendly interface for UNIX and UNIX-like systems. It is being developed by Interactive Systems Corporation, Santa Monica, CA. It is scheduled to be available for the IS/WB around July 1985. Ten Plus provides the user with an object-oriented interface, where each object is a hierarchical data structure. Since Ten Plus appears as a visual editor, objects are acted upon as they would be in a text editor. Objects can be picked up and put down somewhere else, regardless of whether the object is text, a whole file, or a directory tree. Ten Plus will be available on a variety of machines (IBM 43XX, DEC VAX, IBM PC, SCI 1000) so that an organization could have a consistent user interface across the large variety of computing facilities.

7.4.4 UNIX for the IBM PC (PC/IX)

PC/IX is a UNIX-based operating system for the IBM PC. It is developed by Interactive Systems Corporation in Santa Monica, CA. and it provides the same capabilities as the IS/WB for the VAX. Although developed by ISC, it is marketed by IBM corporation. By using PC/IX, a user can perform tasks independent of the VAX CPU and then transfer the work to the VAX using INterm.

7.4.5 INteactive Terminal Emulator (INterm)

INterm is a terminal emulator for the IBM PC and is developed by Interactive Systems Corporation in Santa Monica, CA. INterm allows an IBM PC to emulate various terminal types and to transfer files between the PC and the host. Used in conjunction with PC/IX, development of software can be done on the PC and then transferred to the VAX.

7.4.6 Ada

Since ISSS uses a "plug board" design approach, new tools can be easily added to the system that can use the other features of the environment. An excellent example of this capability would be the addition of an Ada compiler to ISSS. Both the ALS and AIE appear to be delayed for a period of time. By adding an Ada compiler to ISSS, a user gets the advantage of using an advanced programming language and a series of tools to support software development. This allows immediate generation of Ada software combined with the convenience and advantages of the various ISSS tools and utilities.

7.4.7 Profilers

An interesting and useful tool for software enhancement, tuning, and analysis is a profiler. A profiler provides the user with information like the number of times each subroutine is called, the amount of time spent in the subroutine each time, and the total amount of time spent executing particular subroutines. Analysis determines if a bottleneck exists. The bottleneck can be isolated and system performance can be improved. A profiler can also be used to determine the efficiency of various different solutions to a problem. The ideal approach would be to have a language independent profiler, like the VAX has a common debugger. That is, one profiler works with all VAX (or other machine) languages. Such a thing does not currently exist.

7.4.8 Integrated Text and Graphics

A real need in the area of documentation is to automatically integrate text and graphics. A document should be generated in its entirety on one system. The document could contain text, tables, illustrations, photographs, etc. This implies the ability to generate high-quality graphical images on a system in a simple and user-friendly manner. With all aspects of documentation present on one machine, document updates would be substantially easier. There would be no cut and paste operations and losing track of previous versions of the document.

7.4.9 CM of Testing Information

There are various testing tools available currently, but there appears to be no configuration management tool to keep track of the various sets of test data used. There needs to be a tie between a particular module and version of that module and its associated input and output data from the testing procedure. This would improve the quality of the final product and also provide detailed assurance that specific testing guidelines have been met.

8.0 CONCLUSION

General Dynamics developed the Integrated Support Software System to provide AVSAIL with a modern software development and maintenance environment. This final technical report described the ideas, philosophies and concepts involved in its production. It also provides a reflective look to discover what lessons were learned from the project and what direction ISSS modifications should take.

We at General Dynamics look upon the ISSS project as very successful. We see that continual enhancement and standardization is necessary. As developers at both AFWAL and other agencies use the features of ISSS. We hope that its open-ended design will continue to provide many years of productive service.

☆U.S. Government Printing Office: 1985 — 559-065/20821